Feasibility study of a dashboard for extended monitoring of system components in the German satellite data archive.

By
Lorenz Wagner
Student number: 138046
Lorenz.Wagner@et.hs-fulda.de

Hochschule Fulda
University of Applied Sciences
Faculty Electrical Engineering and Information Technology

Bachelor thesis
Industrial engineering, Department of Electrical Engineering

Performed at
Deutsches Zentrum für Luft- und Raumfahrt e.V.
Oberpfaffenhofen, 82234 Wessling

Lecturer
Prof. Dr. Uwe Werner

Mentor
Markus Kunze

Abstract

This bachelor-thesis analyzes the feasibility of a dashboard as monitoring tool for the German National Remote Sensing Data Library (D-SDA). The D-SDA, an archive is operated by the German Remote Sensing Data Center (DFD), part of the German Aerospace Center (DLR) in Oberpfaffenhofen and Neusterlitz.

Firstly, the restrictions of the current systems to monitor the D-SDA over time will be analyzed, so that the requirements for the dashboard can be defined and formulated. Secondly, a prototype will be developed to introduce the design of the dashboard as a monitoring tool for the involved people.

The goal is to analyze the requirements for their feasibility, to recommend if the dashboard would be an useful addition to the current monitoring tools or not.

Acknowledgements

Conducting this research has been a rewarding experience. I would like to use this opportunity to thank these special individuals without whom the completion of this study would have been unimageable.

I wish to thank my supervisor Professor Uwe Werner who assisted me throughout the completion of my research, was subject to dozens of my questions, gave me ideas and challenged my theories. Many thanks!

I further wish to thank my mentor Markus Kunze who was always there for me when I did not know any further or needed guidance and reassurance. The countless times in front of the whiteboard while planning the research helped me to stay focused. You are the best!

I would also like to thank all my interview participants as well as all my colleagues in the Remote Sensing Data Center for their continuing support and for the opportunity to be part of such a well working and organized team.

Finally, I would like to thank my sister Helen Wagner for her caring and support during my thesis and for correcting it. As well as my parents without whom I would not be at this stage of my life.

Content

1. Introduction

1.1. Research Motivation

The monitoring tools currently available to the German National Remote Sensing Data Library (D-SDA) cannot monitor the D-SDA's system component's stability and any occurring errors over time. They only show a snapshot of that very moment they are running. In addition to not having been upgraded for many years, the presentation of the system is also not appealing, making it hard to use. In fact, to observe the current state of a D-SDA's system component, the operator must first search the wanted system component he or she wishes to observe in a list. The information will then be presented as a table within the maintenance program. Having observed this process, it becomes clear that the operators have to invest a lot of effort to get such simple information.

To perform upgrades and to add new features, an external tool would be the best solution. This is because, the current monitoring tools are implemented into the maintenance program and are hard to upgrade. The external tool could display the state and development of the D-SDA over time in a much more effortless style for the operators. Inspired by other projects in the German Aerospace Center (DLR), which uses dashboards for monitoring purposes, a dashboard is considered to complement the current system. Due to its graphical presentation methods, it could display the systems development over time into visual appealing diagrams and provide an addition for the current monitoring tools restrictions. But, the developers in the team being responsible for the D-SDA's constant upgrading to achieve its sustainability work on various projects at the same time. Thus, available time is limited.

To determine whether the benefits of such an external tool would justify the time investment of the developers, responsible clients wish for the projects scope to be investigated. Additionally, the clients want to prevent developing a dashboard that is than not being used for reasons such as, not addressing the needs of the operators.

For the above stated reasons, the projects feasibility is investigated in a feasibility study. To do so this thesis will look into the environment of the monitoring tools and the needs of its users, to then refine the information into requirements. With these requirements the dashboard will be modeled and it´s feasibility will be discussed. In addition, the thesis will describe the development of a prototype. It has the purpose to showcase the comprehensibility of the dashboard's appearance and to test development methods.

1.1. Structure of the document

This thesis firstly introduces the environment of the project to improve the clarity of its context. For this, the DLR, D-SDA and the notion dashboard will be introduced in general.
In the following chapter, theoretical basics of the methods used during the thesis are explained, including requirements engineering as a means to determine the requirements, agile methodologies and Scrum.
Afterwards, the third chapter will demonstrate the process of gathering the required information and outline the methods used during this process. The gathered requirements will be phrased. They are the foundation of the prototype and the feasibility study in general.
In the fourth chapter the process for the planning of the structure of the dashboard will be outlined. It will also model the prototype and investigate the interfaces used for communication. Further, the prototypes structure and development process will be described. In addition, the fourth chapter describes how the finished prototype was presented to the operators and how their impressions were gathered. Due to this feedback the requirements will be adjusted.
Finally, it will be discussed if the dashboard is feasible or not, considering the prior specified requirements. At the end of the chapter, the future of the project is sketched and the question if the dashboard will be developed will be answered.
This is followed by the bibliography and the appendices, including the illustration directory and the list of abbreviations.

1.2. The environment

The DLR is one of Germany's major research facilities. It is specialized in aeronautics, space, energy, transport, digitization and security and also plans the German space program. It has approximately 8000 employees at 20 locations (DLR, 2018). One of its sub institutes is the German Remote Sensing Data Center (DFD) near Munich and Neusterlitz. The DFD maintains national and international receiving stations for satellite data. Its core competences are:

- to operate complex infrastructure at all times,
- to develop IT systems for processing and managing large amounts of data,
- to offer access to remote sensing raw data and earth observation products and
- to safeguard all this data for long term use and following generations.

To manage and store this data the DFD operates a big database called D-SDA with a current capacity of approximately 50 Petabyte. Due to the D-SDA’s mission to safeguard the earth observation data, it is most important to ensure all time stability and functionality. For this reason, the D-SDA and its interfaces are managed by the Data and Information

Management System (DIMS) on the software site. And for redundancy it is distributed to two locations on the hardware site (Dech, 2018).

Image 1: German National Remote Sensing Data Library

Source: Max Wegner DLR Oberpfaffenhofen

The DIMS represents a connection of various individual system components. These system components are hosting different services to simplify the integration process when a new satellite mission is added to the portfolio of the DLR. The DIMS architecture, data integration and workflow are highly customizable to guarantee sustainability. This sustainability is needed to ensure that also in the future, the D-SDA is able to manage the ever-growing satellite data stream. (Molch, 2018)

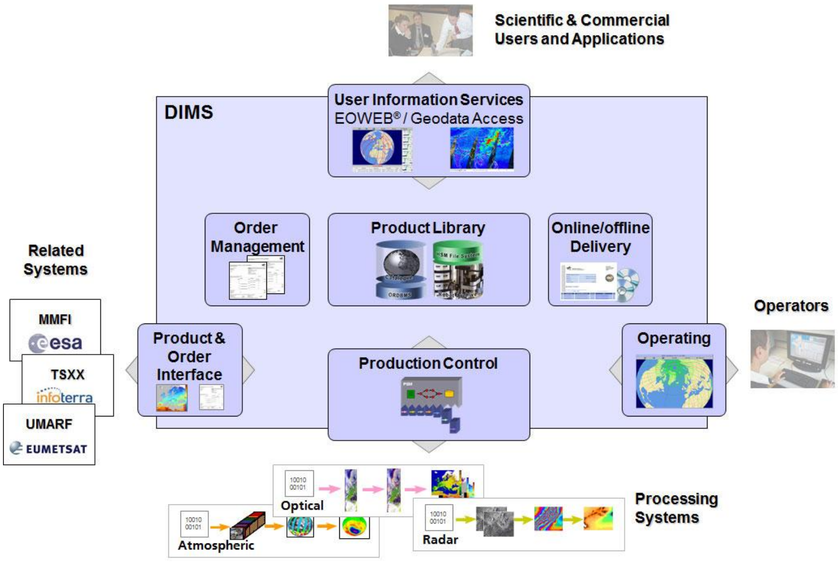

Image 2: Data and Information Management System

(Molch, 2018)

To secure a smooth data exchange between the services, the communication is buffered within queues. The queues are for intermediate storage; their purpose is to prevent that communication is lost when one service works slower or is offline. Sometimes when an error in the DIMS occurs or the system is stressed out, these queues grow tremendous in size. Because of this behavior they are a great indicator to determine the condition of the DIMS.

The DIMS is operated and maintained by the operators. Their daily business is to observe the D-SDAs status, in order to prevent problems and to integrate new data in to the database. They work with a software tool, the already mentioned maintenance program. This program combines the functions of actively configuring and monitoring the DIMS. To utilizing the queues characteristics mentioned above the monitoring tools is able to display the queues state. Nevertheless, as introduced before the possibilities of the current system are restricted. For example, to observe a queue the operator must search for that queue in a list, for it to be displayed in the maintenance tool. The tool showcases the information about the count of entries and their type in a visual unappealing table. However, in a lot of cases, the operators are not interested in the type of data saved in the queue, but rather more in the count. The count is important to, for example, observe the impacts of new mission data added to the DIMS. It showcases the stress level of the DIMS, and its development over time presents a possibility to determine if the level rises or reduces. To keep track of the development the operator currently must memorize the latest states, since only the current state can be displayed. This can make it difficult to observe long-time development of the DIMS performance.

1.3. The dashboard as solution approach

As the current monitoring features are implemented directly into the maintenance program, it is hard to enhance them. Thus, an additional external tool could fill the gap more easily and provide the wanted features. Such a tool could be a dashboard.

Primordial a dashboard was a wood barrier fixed at the front of a horse-drawn carriage. Its purpose was to protect the driver from mud or other debris that were "dashed up" by the horses' hooves. With the development of the first automobiles the notion dashboard moved on. It now describes an array of controls and instruments which has the purpose to present all necessary information in the drivers' field of view (Wiki, 2018).

In business applications, a dashboard also describes a technology that pictures complex information in a graphic based way. Dashboards are often used to display key performance indicators (KPIs) in diagram form to make data more understandable at the first glance (Tricia Aanderud, 2014). In the case of the DIMS, the queues would exactly be such key performance indicators. A major benefit of displaying them in a dashboard would be, that the viewer is able to identify trends more easily, for instance if the queues size is rising or shrinking. And this is exactly what the operators want.

2. Feasibility study

As a dashboard for monitoring purposes in the DIMS has not been evaluated before, risks and gains should be outlined, so that the decision to develop it can be properly founded. These risks and gains could be determined in a feasibility study. A feasibility study typically examines four areas: the technical feasibility, the legal feasibility, the operational feasibility and the scheduling feasibility. A technical feasibility describes a possible solution including the questions, if the proposed system is technical realizable and if the resources to do so are available. The legal feasibility describes, if the proposed system stands in conflict with legal requirements or laws. The operational feasibility describes, how well the proposed system would satisfy the needs for which it is developed. And the scheduling feasibility describes, if a system is able to meet the deadlines set for the project (Matson, 2000). Since the dashboard is only used inside the department and no deadline has been set for its completion, the legal and scheduling feasibilities can be overlooked. In the context of the dashboard, the two remaining feasibilities will be analyzed with the following steps:

- In order to be able to assess the technical feasibility, it is required to investigate what functions the proposed dashboard should include, if it can access the data it needs and if the developers have enough time to develop it.
- To determine the operational feasibility, the prototype will be presented to the operators and their feedback will be gathered, to then discuss if the operators would benefit from a dashboard. In addition, the maintainability of the proposed dashboards will be evaluated to check its sustainability.

However, in order to investigate what functions the proposed dashboard should include and how to develop the prototype, more information on the scope and on the context must be gathered and documented. Based on these documentations, a picture of the system will be modeled. This model will serve as a discussion basis in the feasibility study and showcase the effort needed to develop it, as well as the functionality. The documentation could also serve as a blueprint for the prototype.

The feasibility study is estimated with a duration of two month, including an estimated development time for the prototype of 2 weeks.

2.1. The theory of Requirements Engineering

Requirements engineering (RE) is used to gather and document the information needed for the feasibility study. In brief, RE describes the systematic and scientific journey from an idea, its goals and objectives, and finally the complete definition of requirements. According to the International Requirements Engineering Board (IREB) (Chris Rupp, 2014, p. 13f) the goals of RE are:

- to know the relevant requirements,
- to reach a consensus between the various interested parties about those requirements and
- to document and manage them in a standardized and systematic fashion.

The goals are including to understand and to document the wishes and needs of the different parties involved. These wishes and needs have then to be specified in order to minimize the risk that the end product does not comply with their demands. To document these, it first must be clear who the involved parties are, and then to understand what they really want.

In requirements engineering the parties involved are also called stakeholders. Stakeholders can be all these kinds of persons and institutions, that have an interest or are taking influence on the system. These can be the customers, but also the developers or the audience for a product. As the interests of stakeholders are most important for the success of a project, it is essential to gather and understand the knowledge they provide. The different methods to access this knowledge depend on the type of stakeholder. Most of the time, knowledge is collected with the help of interviews or surveys (Chris Rupp, 2014, p. 80).

In the beginning of a project, stakeholders often do not even know for themselves what they expect of a new system. Thus, it is the task of the requirements engineer to support and to moderate conversations with stakeholders, while documenting their needs and wishes (Chris Rupp, 2014, p. 90f). To prevent misunderstandings in these conversations the engineer must be able to understand what the stakeholders are wanting to express. According to the four-sides model by Schulz von Thun (Image 3), every message can be interpreted with four facets (Chris Rupp, 2014, p. 92).

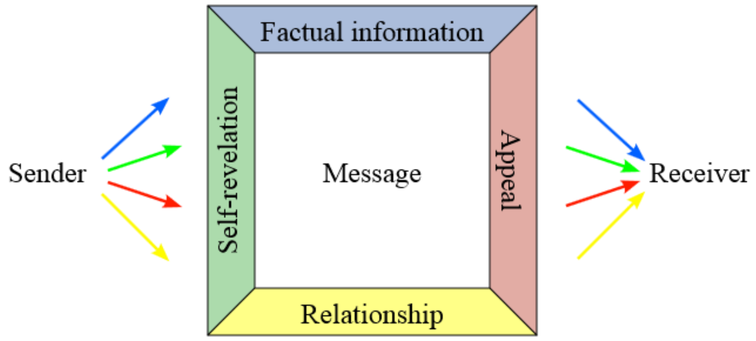

Image 3: The four-sides model by Schulz von Thun

(Chris Rupp, 2014, p. 92)

These facets are:

1) The Factual information facet, in which the facts and information are contained.
2) The Self-revealing facet, in which the sender reveals a part of his or her own mindset, such as his or her feelings.
3) The Relationship facet, in which the sender expresses what he or she thinks about the receiver.
4) And the Appeal facet, which contains the intention of the sender, mostly what he or she wants the receiver to do.

Each of these facets can be misunderstood while gathering information and knowledge. The model shows that communication always happens in multiple layers. In fact, it is not enough to just focus on the Factual information facet, while determining the requirements, otherwise information can be lost (Chris Rupp, 2014, p. 90).

To prevent such communication issues and to also benefit from the small project size, the development team decided, aligning with the operators, to perform the requirements engineering process with an emphasis on agile methodologies.

2.2. Agile process model and documentation in an agile environment

A process model is a methodology, that is used to split processes into small phases. This improves the design and clarity and consequently reduces the complexity (Chris Rupp, 2014, p. 52). Agile as a methodology relies on enhancing the transparency, flexibility and communication in the development process. Following Alistair Cockburn agile implies being effective and maneuverable (Cockburn, 2005, p. 174ff). Furthermore, agile software development is based on four values, which are stated in the Agile Manifesto (Cockburn, 2005, p. 221):

- "Individuals and interactions over processes and tools": The people behind the development process matter more than the tools used. As new solutions are coming to life in discussions, the quality of the interactions between the people matters. This quality is given with uncomplicated and direct face-to-face communication.

- "Working Software over comprehensive documentation": The documentation should be light and meaningful and not hinder the developer when implementing new functions due to its sheer size.
- "Customer collaboration over contract negotiation": When the customer is involved into the development, it is easier for the developer to meet the customer's needs.
- "Responding to change over following a plan": Adapting a plan to match the current situation allows the developers to change priorities to match the customer needs if they change during the development process.

The project's pre-conditions could not match these values any better. The developers sat in the same room and worked with the operators on the same floor. Face-to-face communication was easy to set up, since everyone knows each other personally. Because of the operators' interest in the project, they visited the developers voluntarily to discuss the design and their progress. This also resulted in adapting the plan several times. A lot of the communication happened in front of the whiteboard in the developers' office were the new ideas were written and sketched out immediately. For example, Image 4 shows the first sketched structure of the prototype on the whiteboard.

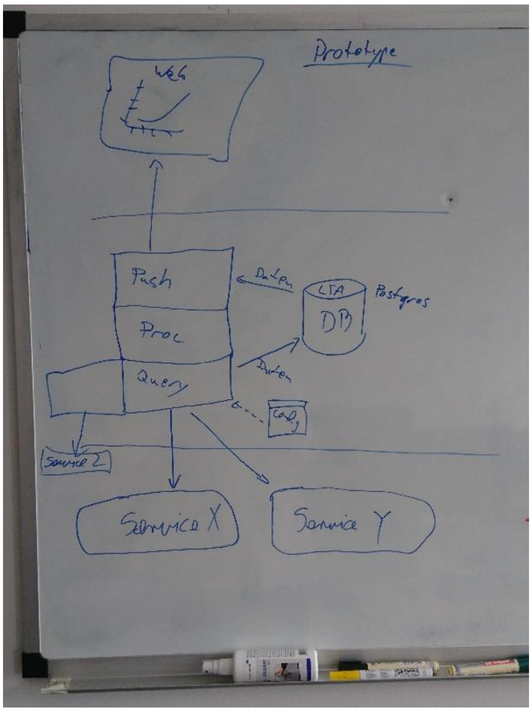

Image 4: Whiteboard

Because produced knowledge can get lost over time, for example through simply forgetting it after a project is finished, through employee fluctuation or through the whiteboard being erased frequently, these ideas have to be documented. As the before mentioned values indicate, agile methods are trying to reduce this documentation to a light but sufficient level. The correct amount of documentation is exactly the amount that is needed for the reader to make the next move (Cockburn, 2005, p. 175). For instance: In agile projects User-Storys are being used to document basic functionalities, whose details are not specified yet. These User-Storys are then used as a discussion basis between the customer and the developers to document the details further during the requirements engineering process.

Often a template is used like: “As <user role> I want the <functionality>, so that <purpose>.” Accordingly, during the whiteboard discussions, the operators’ knowledge and their expectations were gathered in the form of User-Storys (Chris Rupp, 2014, p. 249) to be later refined into requirements.

2.3. Requirements

The goal of requirements engineering is to phrase relevant and qualitatively good requirements. According to the institute of Electrical and Electronics Engineers (IEEE) a requirement is a property or an ability, that is used to achieve a goal or that is needed to comply with a contract or with a standard (Chris Rupp, 2014, p. 13ff).

These requirements can also be a documented representation of such a property or ability. As a consequence, requirements can directly influence the development process. Their purpose is to reflect the understanding and knowledge of all people involved. Through this, they form the vocabulary and therefore are the basis for communication and discussion between the stakeholders. As for documented specifications, requirements are also often the foundation for contracts. Based on their specifications the end product’s quality is tested and evaluated.

Requirements can be separated into two different types: functional requirements and non-functional requirements. A functional requirement is defined as a requirement specifying the result of a behavior provided by a function of the system, according to the IREB (Chris Rupp, 2014, p. 17). Thus, a functional requirement is the demand of a performance the system must provide. A non-functional requirement is every requirement that is not a functional requirement, for example technological, quality or legal requirements. In detail, a non-functional requirement does not only address the system to be developed, but rather the context in which it has been built (Chris Rupp, 2014, p. 268ff).

An example for a requirement could be: A library system shall allow the librarian to enter the bank details of the customer. In addition, this example shows, that requirements are differentiated by their legal liability. Therefore, they are carrying the message of their importance within the fashion they are formulated. Three keywords are used: shall, should and will. If the requirement contains the keyword “shall”, then it must be realized in the finished product. The keyword “should” marks a requirement, that is nice to have but not

necessary to fulfill the contract. And finally, the keyword “will” characterizes an intention, that hints future requirements that have not yet been implemented but are legally binding (Chris Rupp, 2014, p. 18).
To ensure the quality of a phrased requirement, the IEEE defined several quality standards for requirements, listed and described below (IEEE, 1998):

- Completeness: Every requirement must describe a demanded functionality; this also means that the requirement must be measurable.
- Necessity: Every requirement should describe a property, necessary to fulfill the system’s goal.
- Indivisibly: Every phrased requirement should contain only one demand.
- Tractably: Every requirement must be traceable back to its source.
- Technically solution-neutral: Every requirement only describes what is demanded and not how it should be implemented.
- Realizable: It must be possible to realize every requirement under the known circumstances.
- Consistently: Requirements are not allowed to preclude other requirements.
- Explicitly: Every requirement must be understandable in only one manner.
- And testability: Every requirement must be described in a way so that the functionality resulting from it can be tested.

To minimize the expenses of phrasing requirements and to ensure consistent quality, templates are used. The SOPHIST MASTER template (Image 5) (Chris Rupp, 2014, p. 230) delivers a scheme with 6 steps:

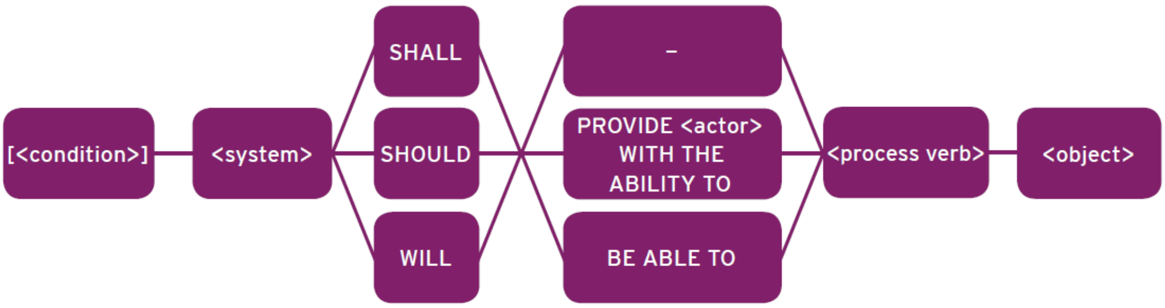

Image 5: SOPHIST MASTER template

(Chris Rupp, 2014, p. 230)

1) The first step is to determine the condition that defines when the functionality is performed.
2) Step two is to identify for which part of the system the functionality is requested.
3) Step three is to set the requirements legal liability.

4) Step four is to determine whether the functionality is a user interaction or for an interface.
5) Step five is to identify the requested functionality.
6) Step six is to identify the object, for which the functionality was demanded.

Besides the quality standards, this template helps to phrase the requirements into a, between all stakeholders shared, vocabulary to help to prevent misunderstandings. Nevertheless, in a written form, requirements can sometimes only describe a graphical user interface or other visual requirements in an inefficient way (Chris Rupp, 2014, p. 286). For this reason and as previously mentioned, a prototype will be developed after the requirements have been documented, to provide a basis for discussions on the design.

2.4. Agile software development and Scrum

This prototype serves as a basis to test if the visualization of the dashboards design matches the operators needs. In addition, it showcases the user interface to give an idea of how the new features could work in the end product. Therefore, a lot communication with the operators is needed, to ensure a design which satisfy their needs. For this purpose and leaning on the agile methodologies, the prototype is developed with the methods of agile software development. Agile software development is a light-weighted approach. As part of it, the requirements and solutions evolve through a collective development and constant communication between the stakeholders. Alistair Cockburn describes the four sweet spots, used to best approach an agile software development project, to benefit from the agile methodology (Cockburn, 2005, p. 178ff). These sweet spots are:

1) Having two to eight people in one room to ensure good communication,
2) using onsite experts for fast feedback,
3) having short development cycles so that priorities can be shifted accordingly, based on the current situation and
4) having experienced developers for efficient development.

To integrate these set of rules a method called Scrum is used. In fact, the development team never used Scrum before and was eager to test it in a real project. Scrum is an iterative process. This means that time-boxed events, so called Sprints, are repeated until the goals are met or the project is canceled. A Sprint is normally 30 days long. And, at its end a potentially deployable product function has been developed. This process is called an incremental process. However, the prototypes Sprint is only two weeks long due to time restrictions. The functions to be developed in such a Sprint are picked from the Product Backlog. This is the list of requirements gathered in the requirements engineering process, sorted by importance and broken down into tasks that can be completed within one day (Schwaber, 2004, p. 142).

Ken Schwaber (Schwaber, 2004, p. 6f) claims that people involved in a Scrum process are organized in three different roles:

1) A Product Owner which is the interface to the customers and the person who is managing the Product Backlog,
2) a ScrumMaster who is responsible for the correct implementation of the Scrum process and
3) the Team which develops the software in every Sprint.

Every Sprint starts with a Sprint planning meeting. During this meeting, the Product Owner presents the Product Backlog items with the highest priority to the Team. The Team then determines how much of these items can developed during the upcoming Sprint. The chosen items are added to the Sprint Backlog, which is essentially a small Product Backlog solely for the upcoming Sprint. In addition, the Team is responsible for organizing themselves and distributing the tasks between each other.

During the Sprint, the Team gathers every day for a short meeting called Daily Scrum. In this meeting Ken Schwaber (Schwaber, 2004, p. 135) argues that everyone answers three questions:

- "What have you done on this project since the last Daily Scrum meeting?",
- "What do you plan on doing on this project between now and the next Daily Scrum meeting?" and
- "What impediments stand in the way of you meeting your commitments to this Sprint and this project?".

The Daily Scrum serves to synchronizes activities and to discuss occurring problems together, to share knowledge and to find the best solution.

At the end of a Sprint a Sprint review meeting is held. Here the Team presents what was developed during the Sprint to the Product Owner and any other further stakeholders interested. In this review meeting the ScrumMaster also discusses with the Team, what can be improved for the future (Schwaber, 2004, p. 9).

3. Requirements engineering process

To gather the information required to assess the scope and the required functionalities of the proposed dashboard, the methods of requirements engineering are used. The goal is to document the operators' needs and to bring these needs into context with each other, so that the scope can be sketched. In addition, requirements are phrased, that can then serve as a discussion basis to investigate the feasibility.

3.1. Stakeholders and interviews

First of all, the relevant stakeholders were identified and documented. Broadly speaking the stakeholders in the proposed dashboards case are divided into the responsible persons as clients, the operators as users and the developers as executers. For a better overview and according to the IREB (Chris Rupp, 2014, p. 81) the stakeholders are all listed in a table. The table contains information about their names, their function, their knowledge regarding the project, their interests regarding the project and their relevance.

Table of stakeholders:

Function	*Name*	*Knowledge*	*Interests*	*Relevance*
Team leader	S.K.	Is familiar with the DIMS.	More efficient monitoring of the DIMS.	Founder
Operator	M.H.	Works with the DIMS.	Easier access to critical runtime information.	Source of information
Operator	J.S.	Operates the DIMS.	Graphical presentation of information in a timeline.	Source of information
Developer	L.W.	Coordinator of the stakeholder inputs.	Analyzing the system.	Requirements engineer.
Senior-developer	M.K.	Oversees the development.	Running end product.	Decision-maker

Table 1: Stakeholders

To access their knowledge, the above mentioned stakeholders were interviewed. They had to answer questions in order for information to be gathered on their relationship with the DIMS, on how they work with the current monitoring system and on what new features they desire. The results of these interviews are presented in form of User-Stories. These stories serve as a discussion basis between the stakeholders to refine the details of the desired requirements.

Team leader:

ID: US01	*Name:* Productivity gains	*Priority:* High
Description: As a team leader I want the application to enable my operators to work more efficiently, so that they can access the necessary data fast and easy.		
ID: US02	*Name:* Prototype	*Priority:* High
Description: As a team leader I want a prototype that showcases the design and logic of the dash-board.		
ID: US03	*Name:* Timely completion	*Priority:* High
Description: As a team leader I want the feasibility study including the prototype to be finished by the end of March, so that it is finished before the developer's internship ends.		
ID: US04	*Name:* Prototype reusable	*Priority:* Moderate
Description: As a team leader I want the prototype to be reusable for the development of the End-Product.		
ID: US05	*Name:* Robust system	*Priority:* High
Description: As a team leader I want the system to be designed as robust as possible.		
ID: US06	*Name:* Predictable system load	*Priority:* Moderate
Description: As a team leader I want the system load generated by the monitoring to be predictable.		
ID: US07	*Name:* Multiple distribution	*Priority:* Moderate
Description: As a team leader I want the dashboard to be accessible by multiple operators at the same time.		
ID: US08	*Name:* Interfaces	*Priority:* High
Description: As a team leader I want the dashboard to use already existing interfaces.		

Table 2: User stories - Team leader

Operators:

ID: US09	*Name:* Appealing presentation	*Priority:* High
Description: As an operator I want the queues to be displayed in a diagram over time, so that I can identify problems at first sight.		
ID: US10	*Name:* Display customizable	*Priority:* Moderate
Description: As an operator I want to choose which ques are to be displayed, so that I can concentrate on the ques I need for my work.		
ID: US11	*Name:* Timeline customizable	*Priority:* Moderate
Description: As an operator I want to choose the diagrams maximal length in time.		
ID: US12	*Name:* Count customizable	*Priority:* Moderate
Description: As an operator I want to choose which and how many queues are displayed.		
ID: US13	*Name:* Timeframe	*Priority:* High
Description: As an operator I want the diagram to be able to display up to 7 days of old data.		

Table 3: User stories - Operators

Developers:

ID: US14	*Name:* Agile process model	*Priority:* Low
Description: As a developer I want to develop with agile methods, so that I can learn more about this process model and the methods of Scrum.		
ID: US15	*Name:* Documentation	*Priority:* High
Description: As a developer I want to create a meaningful documentation, so that my successors can easily edit and further develop the prototype.		

Table 4: User stories - Developers

Out of the user stories, the main goals of the project can be derived. The goals illustrate the orientation and help to imagine the desired end state. First, the system must allow a more efficient workflow. Second, the output must be graphical and customizable. And thirdly, the system must be robust. To document the goals and to provide additional information they are noted in a template (Chris Rupp, 2007, p. 188ff).

Main goals:

Goal: More efficient workflow.
Stakeholder: Team leader, Operators.
Effect on the Stakeholder: Less effort to monitor the system.
Restrictions: The system should be resource conserving and thus, the presentation must be light.

Table 5: Goal one - Efficiency

Goal: Graphical and customizable display.
Stakeholder: Operators.
Effect on the Stakeholder: The operators can alter the design in favor of their needs.
Restrictions: None.

Table 6: Goal two - Customizable

Goal: Robust and fail-safe system.
Stakeholder: Team leader.
Effect on the Stakeholder: The system should be reliable to be able to monitor even if problems occur.
Restrictions: The system load should be predictable.

Table 7: Goal three - Robust

3.2. Context of the project

As the new features will be implemented into an already running environment, it is important to mark out the monitoring system's boundaries and interfaces.

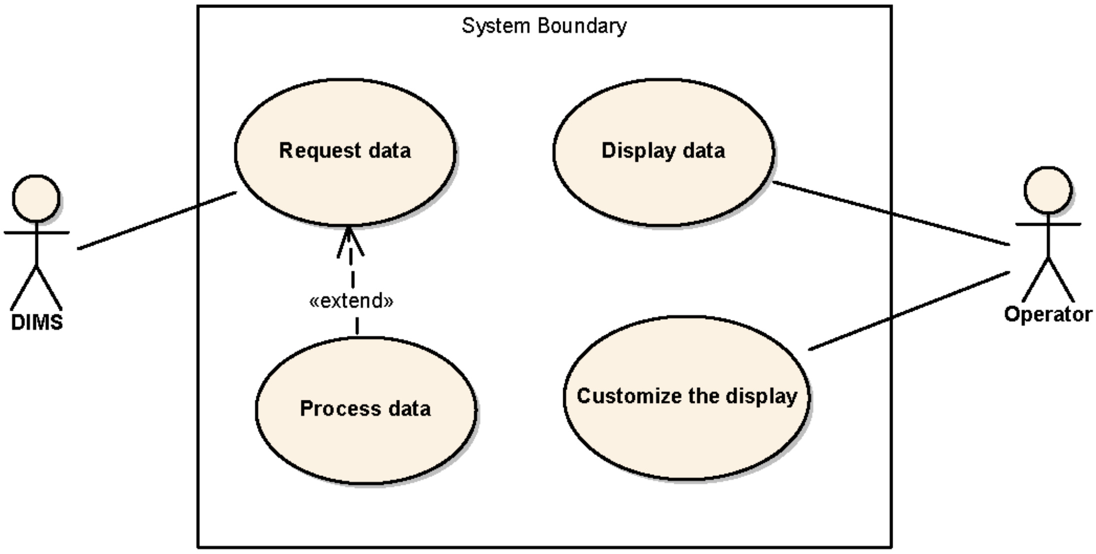

Image 6: System context diagram

As displayed in image 6, there are two major interfaces with the surrounding. Within the first interface the relevant data from the DIMS is requested. For example the status of the queues or if a service is online or offline. The second interface will display the processed data from the DIMS to the operators, in a graphical form. In addition, it will give the operators the possibility to individualize the display. Since there can be more than one operator logged into the system simultaneously, the system must be able to distribute the data out of one DIMS interface into multiple presentation interfaces. The interfaces practical structure and realization will be discussed more closely during the development of the prototype.

On the other hand, the operator will neither be able to edit the data nor to access the unprocessed DIMS data or the DIMS in any manner. The system's only purpose is to automatically display and refresh the data in a visual appealing design.

3.3. Use Cases

Since user stories mostly focus on what a stakeholder is demanding, they are great to consider the priorities of the different functionalities. However, in doing so, they are leaking the details on how to realize their subject. In addition, they don't show their relation to other user stories (Chris Rupp, 2014, p. 249).

To give an overview about the whole system and its surroundings, the user stories were refined in discussions with the stakeholders. A use case diagram was created to showcase the system's desired functions as individual use cases and to extend them with their main features (Image 7).

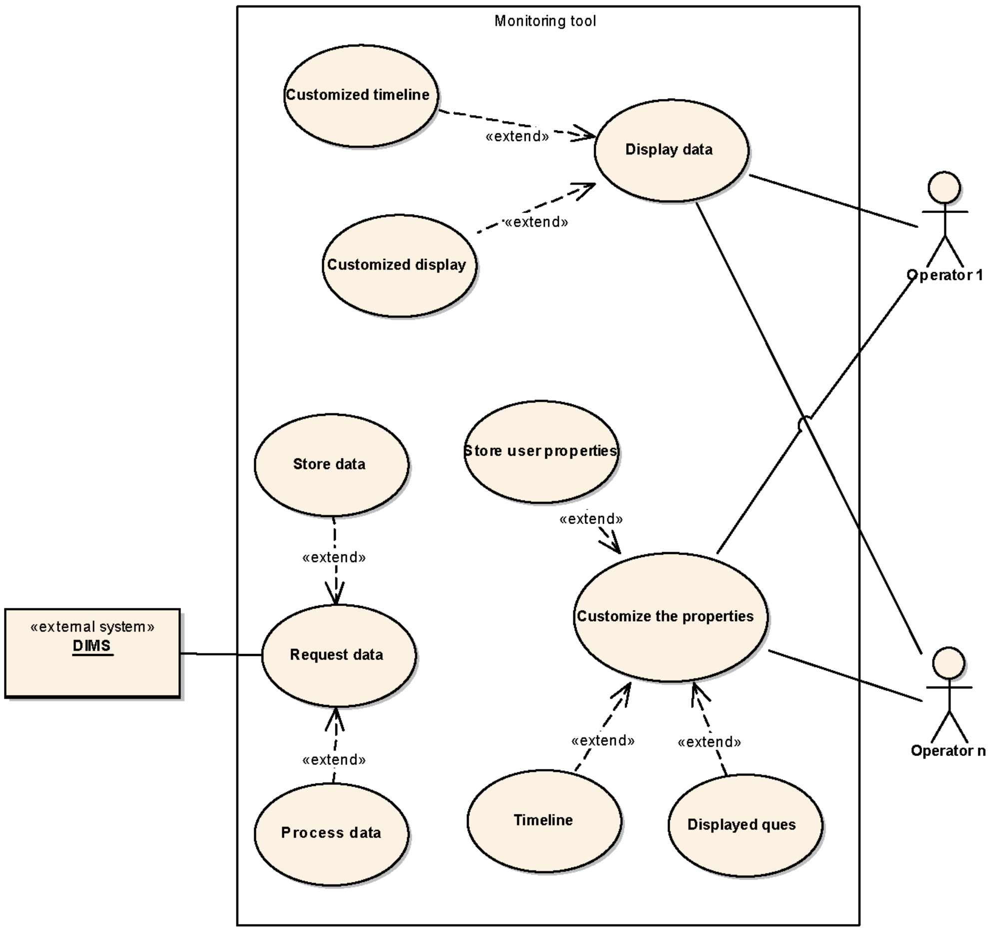

Image 7: Use case diagram

As the use case diagram depicts (Image 7), there are three major functionalities demanded for the proposed dashboard:

1) The "Request data",
2) the "Customize the properties" and
3) the "Display data" use case.

Every single one of the use cases describes an interaction with another person or system. Use cases are extended by their dependencies within the monitoring tool. They are great in outlining the dashboards planed functionality: The monitoring tool is requesting data through the request data use case from the DIMS, processes and stores this data and then presents it in the display data use case, in the operators customized design.

To bring the functionalities into the context with the non-functional requirements and to showcase the procedure, they are described more closely in the following use case descriptions. For this reason, the use cases are noted in tables depicting important background information about their function and answering the questions, i.e. what the use cases are doing, when they are activated and what the result is.

Use case description:

Use case name: Request data
Description: The monitoring system requests data from the DIMS which is relevant for its purpose.
Involved systems and stakeholders: The DIMS, the monitoring system.
Preconditions: The two systems are already connected.
Result: The monitoring system is pulling data from the DIMS.
Trigger: A predefined timer.
Procedure: 1. The timer fires. 2. The monitoring system is sending a query. 3. The DIMS allocates the data. 4. The monitoring system pulls the data. 5. The monitoring system processes the data and stores it.
Technical requirements: The monitoring system only queries the DIMS once per logged in operator after a predefined timer fires to ensure low system load. The monitoring system only requests data that is currently displayed by an operator to conserve resources. The monitoring stores the data internally, so that it does only need query the DIMS once independent of how many operators are logged in.

Table 8: Use case one - Request data

Use case name: Display data
Description: The monitoring system displays data.
Involved systems and stakeholders: The monitoring system, the operators.
Preconditions: The operator is logged in.
Result: The monitoring system is displaying the data in form of a chart on the operators screen.
Trigger: The operators' defined refreshing rate.
Procedure: 1. The refreshing rate timer fires. 2. The monitoring system is plotting the new data.
Technical requirements: The monitoring system can handle multiple connected operators.

Table 9: Use case two - Display data

Use case name: Customize the properties
Description: The operators can customize their properties.
Involved systems and stakeholders: The monitoring system, the operators.
Preconditions: The operator is logged in.
Result: The monitoring system stores the operator's properties for "which ques are being displayed" and "what is the refreshing rate and the timelines scope".
Trigger: The operator is changing a property.
Procedure: 1. The operator changes a property. 2. The monitoring system is storing the change.
Technical requirements: The monitoring system stores the style of every operator.

Table 10: Use case three - Customize the properties

3.4. Requirements

Based on the use cases and the user stories the requirements are documented. They are the foundation to assess the feasibility and are the blueprint to develop the proposed dashboard, as well as the prototype. Every requirement addresses one functionality and the priority described in the use cases is translated into legal liability. They are divided into functional and non-functional requirements.

The requirements are phrased with the help of the SOPHIST MASTER template described before. To reference these during the development process and in the discussion of the feasibility study, they are identifiable with unique IDs.

Functional requirements for the system:

F-Dash-001:
When an operator starts the system, it shall identify the connecting operator.

F-Dash-002:
When an operator starts the system, it shall automatically display the data in a diagram with a time axis.

F-Dash-003:
When an operator starts the system, it shall load the operators saved customized design.

F-Dash-004:
If an operator wants to alter his design, the system shall provide the operator with the ability to customize it.

F-Dash-005:
If an operator customized his display, the system shall save these changes.

F-Dash-006:
The system shall be able to customize the time scale of the diagrams axis.

F-Dash-007:
If the time scale of the diagrams axis is customized, the system shall be able to display up to 7 days of old data.

F-Dash-008:
The system shall be able to display multiple diagrams at the same time.

F-Dash-009:
The system shall be able to display different data sources in different diagrams.

F-Dash-010:
The system shall provide the operator with the ability to customize the count of different diagrams displayed.

F-Dash-011:
The system shall provide the operator with the ability to swap the data source shown in the different diagrams.

F-Dash-012:
The system should be able to distribute data to multiple operators.

F-Dash-013:
If multiple operators log into the system, it should be able to display every operator his or her custom design.

F-Dash-014:
If the system is connected to the DIMS, the system’s load should be predictable.

Non-functional requirements for the system:

NF-Dash-001:
The diagram shall be designed in a way so that the information it contains is understandable and feasible.

NF-Dash-002:
The system should be designed in a way that already existing interfaces can be reused.

NF-Dash-003:
The system should be designed in a way to be robust.

NF-Dash-004:
If connection issues occur, the system shall continue to work and display that an error occurred.

NF-Dash-005:
If any part in the system needs longer to process a work step, the system shall display the delayed data at the correct position in the timeline.

NF-Dash-006:
The system should be developed with an agile process model and Scrum.

NF-Dash-007:
The systems development shall be documented.

Non-functional requirements for the prototype:

NF-Proto-001:
The prototype shall be developed in a way, that displays the dashboards design.

NF-Proto-002:
The prototype should be reusable in the development of the dashboard.

NF-Proto-003:
The prototype shall be developed until the end of March.

4. The prototype

In addition to the requirements, the prototype is developed to, be able to analyze the operational feasibility. Therefore, it should give an overview about the design in order to, test the operator acceptance for the new tool in the end. In contrast to the proposed dashboard, the prototype does not need to include all the previously gathered requirements. It is therefore important to decide which requirements are to be implemented. To manage this process and the development, a method called Scrum is used. The theory of Scrum was already explained in the second chapter. Therefore, this chapter will only deal with its practical implementation in the development.

Particularly important in the prototypes development, is to ensure that the design matches the needs of the operators. Scrum will implement the operators through the Product Owner role into the development, which allows the developers to be in constant contact to them, and to gain regular feedback on their design choices. Due to time restrictions, the prototype is developed in only one Sprint with a duration of 2 weeks. Nonetheless, this Sprint is planned in a Sprint planning meeting in which the procedure for the Sprint is discussed. The Team formed by the senior developer and the junior developer, organized itself in a way so that the senior developer provided guidance to the junior developer rather than actively developing himself; the junior developer developed the prototype alone. Due to the fact, that both worked in the same office, communication was rather uncomplicated and the junior developers' questions were immediately answered. In addition to the organization, the developers as Team and the operators as Product Owners discussed together, which requirements to add to the Sprint Backlog (Image 8). Besides the special prototype requirements, every requirement affecting the design was considered viable and added.

To document them, they were pined on the Backlog. It contains the requirements:

- *F-Dash-001*
- *F-Dash-002*
- *F-Dash-004*
- *F-Dash-005*
- *F-Dash-006*
- *F-Dash-008*
- *F-Dash-010*
- *F-Dash-012*
- *NF-Dash-001*
- *NF-Dash-006*
- *NF-Dash-007*

- *NF-Proto-001*
- *NF-Proto-002*
- *NF-Proto-003*

These requirements were then divided into tasks (Image 8), which could be realized in a day's work.

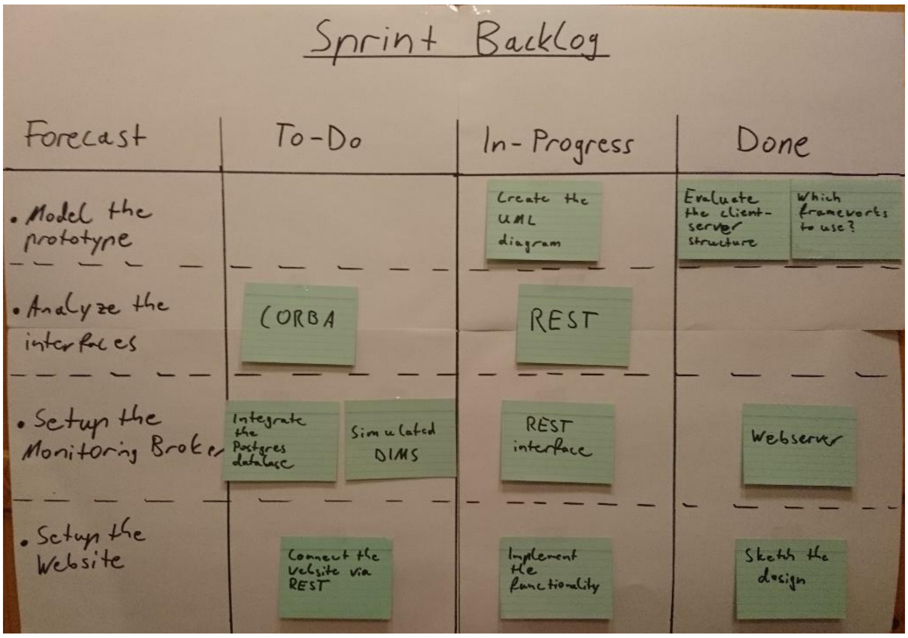

Image 8: The Sprint Backlog

Since only the junior developer is actively developing, every task is assigned to him.
During the development process the Daily Scrum was taken place in the developers' office. Because the operators work on the same floor, they visited frequently. The operators could see and review the design progress during the daily Scrums presentation of the work done on the day before and could give immediate feedback to the junior developer. This proved to be essential during the design process, since the design evolved through these discussions to match the operator's needs.
Wrapping things up, after the development was finished, the prototype has been presented to the operators in the Sprint Review Meeting. Because of its importance the review meeting is described more closely at the end of this chapter.

4.1. Modelling

The first task in the Sprint Backlog is to model the proposed dashboard's structure. The use case diagram and goals from the requirements engineering process are already hinting at the desired structure. Data should be read from the database, processed and saved to then be displayed in a diagram. This structure now must be refined considering the requirements inside the sprint backlog.

In order to implement the best structure during development, the developers needed to decide which program languages to use and how to implement the different functionalities. Considering the requirement *F-Dash-012* it would be beneficial to split the proposed dashboard into two parts. First, the visualizing part which runs on the operators' computer and displays the diagrams. And second, the data providing part which runs in the DLR network and distributes data to multiple operators. Such a solution would also meet the requirement *F-Dash-014*, since the operators would receive their data from the data providing part. This part needs to query the DIMS and thus, the system load would be predictable. In addition, the data providing part could host a database, which would enable it to save the data queried from the DIMS (*F-Dash-007*) and to save the customized design from the operators (*F-Dash-005*).

The visualizing part of the proposed dashboard would be displaying the data in graphical form (*F-Dash-002*) and accepting the user input to customize the design (*F-Dash-004*). Since the junior developer worked with website development in his internship prior to the feasibility study, the two developers decided to realize the visualizing part in form of a website. A website could display the data in graphical form, for instance as diagrams, without too much development effort (*F-Dash-006, F-Dash-008, F-Dash-010*). In addition, it could run on every internet browser. This means, that no extra application would be required, and the dashboard could be started through a Hyperlink.

To document the proposed model and to clarify the connections between the parts and their components, a UML component diagram is used. UML is the abbreviation of Unified Modeling Language. It is a general-purpose graphical modeling language in software engineering. With it, a blueprint for the development is created (Siegel, 2005).

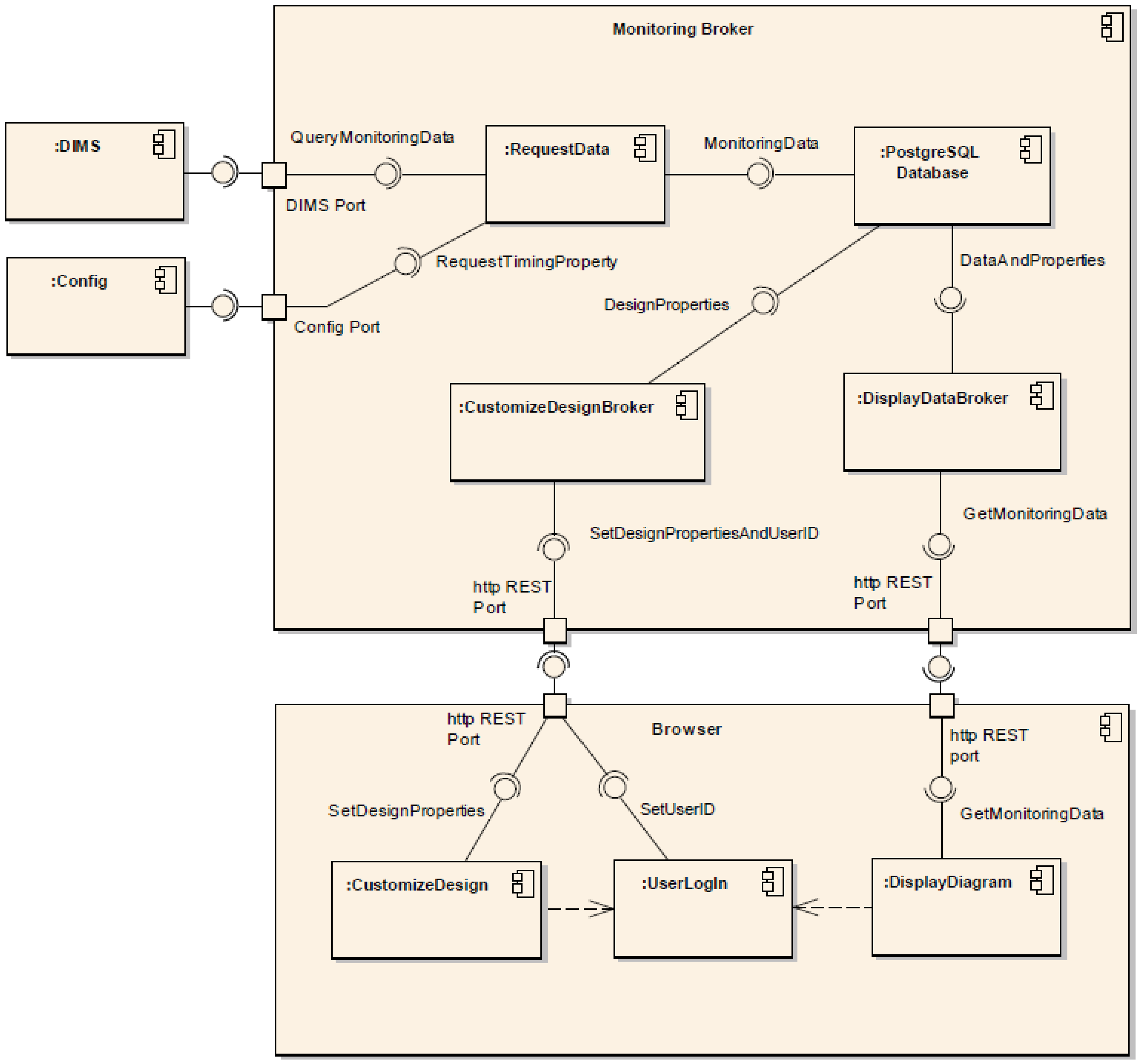

Image 9: Prototypes component diagram

The component diagram brings the prior described structure into a graphical form. Here the proposed dashboard is splited into the two parts. The visualizing part as client site is labeled as Browser in the diagram, it represents the website that visualizes the data. And secondly, the data providing part as server is labeled as Monitoring Broker, querying the data from the DIMS, saving it in its own database and distributing it to the different clients. They are connected trough an interface which is called “http REST Port” as par the diagram. REST will be explained in the next chapter about interfaces. The second interface the Monitoring Broker uses, connects it with the DIMS, however in the prototypes case the DIMS is only simulated. The purpose of the external config file is to simplify the change of the properties without expert knowledge.

The two main components are further divided into smaller classes which are representing the internal structure. In fact, the Monitoring Brokers component is grouped into three classes and the integrated database. The three classes are:

1) The RequestData class which queries data from the DIMS in periodical intervals. These intervals are set by the RequestTimingProperty read from an external Config file. In the prototypes case the DIMS is simulated by an external Plugin which provides random numbers. The MonitoringData from the RequestData class is then saved data in the database.
2) The CustomizeDesignBroker class retrieves the design properties and the client ID from the client through the interface. It saves the data into the database.
3) The DisplayDataBroker class reads the MonitoringData and the DesignProperties from the database and provides it through the interface to the client.

The Browser component also contains three classes:

1) The DisplayDiagram class displays the graphical output based on the data received from the Monitoring Broker.
2) The UserLogIn class identifies the current user and provides its ID to the Monitoring Broker.
3) And the CustomizeDesign class will provide the input options to the user and sends the inputs trough the interface.

Since the prototype is planned as a simplified version of the proposed dashboard, the prototype simulates the DIMS as an external class with a Plugin. This Plugin can later be swapped for others, like an interface Plugin to communicate with the DIMS. As result, the prototype could be completely reused in the proposed dashboards development, so that the requirements *NF-Proto-001, NF-Proto-002* are fulfilled.

4.2. The dashboards two interfaces

Since the communication between the components is most important for their functionality, they are connected through interfaces, enabling the components to communicate with each other. To simplify the data exchange interfaces are commonly standardized. Such a standardized interface is also called an application programming interface (API). An API is a set of definitions that describes the interface, that an application is using. It is a set of methods to clearly define the communication between components and thus, making it easier for the developers to establish communication. The major benefit is that APIs simplify communication, so that the developer can use predefined functions without requiring, that he or she understands the underlying logic of an API (Clarke, 2014).

In the proposed dashboard's case, the connection between the DIMS and the Monitoring Broker is based on a standard called common object request broker architecture (CORBA). CORBA is specialized to enable connection between diverse platforms, like different operating systems or different programming languages, on a single system or in networks. It is essentially a design blueprint for an Object Request Broker (ORB). Trough such an ORB, distributed objects can communicate with each other and even call functions from one another. In practice, if an object wants to call a function in another object, it wraps the function's call in a specific way and sends it to the broker. The broker than sends the wrap the other object where it is unwrapped and processed. The results are again wrapped and sent through the broker back to the origin object. The way the call must be wrapped is defined with the Interface Definition Language (IDL). In it, the structure of the objects interfaces is described. CORBA is the DIMS' main communication protocol it not only controls the communication with the monitoring tools but between the DIMS services itself (OMG, 2012).

The second standard used is called Representational State Transfer (REST). It connects the Monitoring Broker with one ore multiple browsers. REST is a common protocol in the internet communication which is specialized on machine to machine communication. A REST interface has a unique address in the network or the internet called Uniform Resource Locator (URL), which is an address of a website. RESTs communication is stateless which means, that each request from client to server must contain all necessary information to process the request. In the prototype's case the REST interface displays the data in the form of the JavaScript Object Notation (JSON), which is essentially plain text. As example image 10 shows the monitoring data which is provided by the Monitoring Broker for the browser. It is expressed in an Array which contains an id, the queues id, a timestamp as well as the value for that timestamp (Fielding, 2000).


```
{"id":1,"que":1,"value":
[42,34,34,26,21,29,38,31,2,7,
```


Image 10: Prototypes REST interface

4.3. The prototype

The prototype was developed based on the modeled blueprint. To realize the different functions various technologies were used. The Monitoring Broker is mainly programmed with the program language Java. Java is used because the CORBA implementation of other already in the DIMS working java applications, could be reused to simplify the development. Besides, a framework named Spring is used providing a broad number of predefined functionalities (Phillip Webb, 2018).

In the prototype’s case the framework Spring will host the website, it will provide the server site of the REST interface and it will integrate the database into the Monitoring Broker. For the database the prototype will use PostgresSQL. Postgres is an easy scalable database which is an open source with a 15-year history. It is free and contains more functionalities than required, meaning that the database will not limit further upgrades of the dashboard (Postgres, 2018).

As the prototype is not connected to the DIMS, the incoming data must be simulated. To generate data the prototype will generate random values between 0 and 50 every second. The values are saved together with a unique id, the queue they belong to and a timestamp into the database. In addition, the database saves the design properties for the customized design. The database’s data can be accessed via the REST interface.

When a user opens the prototypes website, the browser requests the design properties and the values from the database. As shown in image 11 it displays the data by default in form of 4 diagrams showing 50 values each. On the left side these design properties can be adjusted through sliders. They allow to display one to four diagrams with 20 to 100 values each. If the user adjusts the settings, the changes will be saved inside the database. The diagrams will refresh automatically every 5 seconds to display the new data. The line diagrams show the value of the monitored queue on the y-axis and the development of the values over time on the x-axis.

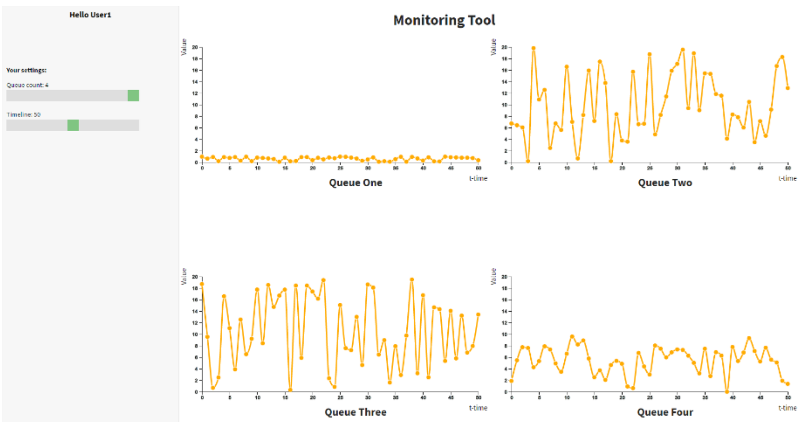

Image 11: The prototype

4.4. New Requirements

As the final step, the prototype has been presented to the operators. In a small conference room, the developers met with the operators to demonstrate the prototype's design and its functions. After the operators tested it for a while, the group started to discuss the ups and downs of a dashboard as an additional monitoring tool.

Overall the operators' reaction on the prototype was quite positive. They liked the idea of visualizing the queues status in form of a line diagram, because it gives a clear understanding of the past development and the future trend. Secondly, they enjoyed the simplicity of the dashboard as a browser application. They can use the browser they like most and do not have to install another desktop application. The prototype can be easily started by opening the dashboard's website. Finally, they immediately started and customized the design of the prototype, the operators liked that they are able to give their dashboard a personal touch.

Nevertheless, they also gave some constructive feedback. They wished, as a new design feature, that the diagram should display more clearly when it refreshes, so they would immediately notice any changes of the queues status, because the change would be more eye-catching. They also wished to see additional information on the data inside the queue in an extra table, for example the satellite it comes from or what type of data it is. And, they wished to also have an all-time statistic about the data saved inside the D-SDA, such as the amount of data saved, the number of different data packages saved and how many of these data packages belong to which satellite mission.

Since the refresh animation is just about the design and the CORBA interface already provides the additional information of the queues content, both wishes are easy to implement and could be added as requirements into the Product Backlog in the next Sprint Review meeting. The third wish about the all-time statistic is, however, not as easy to implement. To display the statistic, the dashboard would require a lot of data from the DIMS every time an operator starts it, which would stress the interface and therefore would not comply with the requirement about the predictable system load. To realize it, further research is needed. However, as it would be a complete new functionality, this is no longer part of the feasibility study.

Thanks to the iterative design of Scrum, the operators in the role of the Product Owners, are able to add their feedback into the Product Backlog. Thus, the two new requirements are:

F-Dash-015:
If the diagram refreshes it data, it shall animate the process so that the operator notices the change.

F-Dash-016:
If the operator activates the function, the dashboard shall display an extra table containing additional information about the data that is currently buffered inside the queue.

5. Summary of the feasibility study

The main objective of this thesis is to assess the feasibility of a dashboard for extended monitoring purposes. For this reason, the current system has been described and its restrictions have been outlined. Based on these restrictions, information was gathered to be able to determine which functionalities the dashboard would require to extend the monitoring. The needed functionalities were documented in form of requirements to serve as a blueprint for the dashboard and to determine its scope. Afterwards, a prototype has been developed and presented to showcase how the dashboard could function and to be able to discuss if the design matches the need to present data in an easy and understandable way.

As chapter two indicates, the study should answer the technical and operational feasibility. To investigate the technical feasibility, the prototype contained all the major functionalities the dashboard would require, such as the client – server communication, the integrated database and the visualization. In addition, the interfaces have been described in chapter four and this showed, that they can be easily implemented into the dashboard. As a result, it can be said that the dashboard is technical feasible. To investigate the operational feasibility, the proposed dashboard was designed to be sustainable through the usage of open source technology that is also very scalable and thus, new features could be added in the future. Its usability has been discussed during the presentation of the prototype. It became clear that the operators liked the dashboard as a monitoring tool and they immediately started to talk about what possibly could be done with it through adding new features. They liked the easy design which allows them to perceive the queues development at a glance. This helps the operators to save time during their work and thus, makes the monitoring more efficient which is beneficial for their work.

On the other hand of all these benefits, the proposed dashboard has to be developed. Since the time of the developers being responsible for the D-SDA's development is limited, they are the crucial factor. In its current state the proposed dashboards only purpose is to simplify the monitoring of the queues. Measured against the complexity of the whole DIMS, this is not much. Nevertheless, based on its on sustainability focused design, the dashboard could serve as a vessel for further upgrades of the monitoring system.

To summarize, the study comes to the result that a dashboard as monitoring tool is beneficial for the operators' work, but the resources for its development are limited. Since the current monitoring restrictions are no urgent problems, the proposed dashboard must not be realized straight away. Considering this, the dashboard is an investment which will pay off when viewed on long-term, and could still be realized when the developers have time to spare.

6. Bibliography

Chris Rupp, S., 2007. *Requirements -Engineering und -Management, Professionelle, iterative Anforderungsanalyse für die Praxis.* 4th ed. Nürnberg: Carl Hanser Verlag München Wien.

Chris Rupp, S., 2014. *Requiremetns -Engineerung und -Management, Aus der Praxis von klassisch bis agil.* 6th ed. Nürnberg: Carl Hanser Verlag München.

Clarke, S., 2014. *drdobbs.* [Online]
Available at: http://www.drdobbs.com/windows/measuring-api-usability/184405654
[Accessed 08 04 2018].

Cockburn, A., 2005. *Agile Software Development.* 8th ed. Boston: Pearson Education, Inc..

Dech, P. S., 2018. *The German Remote Sensing Data Center (DFD).* [Online]
Available at: http://www.dlr.de/eoc/en/desktopdefault.aspx/tabid-5278/8856_read-15911/
[Accessed 27 03 2018].

DLR, 2018. *DLR at a glance.* [Online]
Available at: http://www.dlr.de/dlr/en/desktopdefault.aspx/tabid-10443/637_read-251/#/gallery/8570
[Accessed 27 03 2018].

Fielding, R. T., 2000. *Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation.* Irvine: University of California.

IEEE, I. o. E. &. E. E., 1998. *830-1998 - IEEE Recommended Practice for Software Requirements Specifications,* s.l.: IEEE.

Matson, J., 2000. *Cooperative Feasibility Study Guide,* Washington: United States Department of Agriculture.

Molch, K., 2018. *Team: D-SDA Services.* [Online]
Available at: http://www.dlr.de/eoc/en/desktopdefault.aspx/tabid-7425/12496_read-30567/
[Accessed 27 03 2018].

OMG, O. M. G., 2012. *Common Object Request Broker Architecture (CORBA),* Needham: Object Management Group.

Phillip Webb, D. S. J. L. S. N. R. W. A. W. M. O. C. D. S. D. M. S. V. P. J. B. M. B., 2018. *Spring Boot Reference Guide.* [Online]
Available at: https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/
[Accessed 08 04 2018].

Postgres, T. P. G. D. G., 2018. *About.* [Online]
Available at: https://www.postgresql.org/about/
[Accessed 08 04 2018].

Schwaber, K., 2004. *Agile Project Management with Scrum.* 1th ed. Redmond: Microsoft Press.

Siegel, D. J., 2005. *Introduction To OMG's Unified Modeling Language.* [Online]
Available at: http://www.uml.org/what-is-uml.htm
[Accessed 08 04 2018].

Tricia Aanderud, A. D. I. M. H. M., 2014. *Dashboards with SAS® Visual Analytics,* Cary (North Carolina): SAS Institute Inc..

Wiki, 2018. *Wiki Dashboard.* [Online]
Available at: https://en.wikipedia.org/wiki/Dashboard
[Accessed 27 03 2018].

7. Illustration directory

8. Table directory

9. List of abbreviations

API	Application programming interface
CORBA	Common Object Request Broker Architecture
D-SDA	German Satellite Data Archive
DFD	German Remote Sensing Data Center
DIMS	Data and Information Management System
DLR	German Aerospace Center
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IREB	International Requirements Engineering Board
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
ORB	Object Request Broker
RE	Requirements engineering
REST	Representational State Transfer
URL	Uniform Resource Locator

10. Appendices

Explanation of self-sufficiency

I insure that I composed this written performance independently and that I did not use any other than the stated sources. I also insure that I marked every spot with sources in the text where I cited other works verbally or in terms of. This work is not uses for other performances before.

Ich versichere, dass ich die vorliegende schriftliche Prüfungsleistung selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe und die Stellen, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, im Text jeweils mit Quellenbelegen kenntlich gemacht habe. Die Arbeit ist noch nicht anderweitig für Prüfungszwecke vorgelegt worden.

Bad Orb, 08.04.2018

(Ort, Datum/Place, Date) (Unterschrift/Signature)